European Journal of Operational Research 234 (2014) 61–76

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A modified variable neighborhood search for the discrete ordered median problem

Justo Puerto [a,*], Dionisio Pérez-Brito [b], Carlos G. García-González [c]

[a] Dpto. de Estadística, Investigación Operativa Universidad de Sevilla, Spain
[b] Dpto. de Estadística, Investigación Operativa y Computación Universidad de La Laguna, Spain
[c] Dpto. de Economía de las Instituciones, Estadística Económica y Econometría, Universidad de La Laguna, Spain

ARTICLE INFO

Article history:
Received 17 November 2012
Accepted 18 September 2013
Available online 4 October 2013

Keywords:
Discrete ordered median problem
Variable neighborhood search
Discrete facility location

ABSTRACT

This paper presents a modified Variable Neighborhood Search (VNS) heuristic algorithm for solving the Discrete Ordered Median Problem (DOMP). This heuristic is based on new neighborhoods' structures that allow an efficient encoding of the solutions of the DOMP avoiding sorting in the evaluation of the objective function at each considered solution. The algorithm is based on a data structure, computed in preprocessing, that organizes the minimal necessary information to update and evaluate solutions in linear time without sorting. In order to investigate the performance, the new algorithm is compared with other heuristic algorithms previously available in the literature for solving DOMP. We report on some computational experiments based on the well-known *N-median* instances of the ORLIB with up to 900 nodes. The obtained results are comparable or superior to existing algorithms in the literature, both in running times and number of best solutions found.

1. Introduction

Location analysis is a very active topic within the Operations Research community. It has given rise to a number of nowadays standard optimization problems some of which are in the core of modern mathematical programming. One of its more important branches is Discrete Location. Witnesses of its importance are a number of survey articles and textbooks that collect a large number of references on methodological results and applications, see e.g. Daskin (1995), Drezner and Hamacher (2002), Mirchandani and Francis (1990), Nickel and Puerto (2005) and references therein. Roughly speaking, Discrete Location problems typically involve a finite set of *sites* at which facilities can be located, and a finite set of *clients*, whose demands have to be satisfied from the facilities.

An important aspect of a location model is the right choice of the objective function and in most classical location models the objective function is the main differentiator. Therefore, a great variety of objective functions has been considered.

Discrete Ordered Median Problem was introduced to provide a unifying way to model many location models see e.g. Nickel (2001), Boland, Domínguez-Marín, Nickel, and Puerto (2006) and Nickel and Puerto (2005). It has been recognized as a powerful tool from a modeling point of view because it generalizes the most popular objective functions in the literature of location analysis and it also allows to distinguish the different roles played by the different parties in a supply chain network. The correct identification of the different roles played by the agents participating in logistics models has led to describe and analyze new types of distribution patterns, namely customer-oriented, supplier-oriented or third party logistics provider-oriented, see (Kalcsics, Nickel, Puerto, & Rodríguez-Chía, 2010a, 2010b; Puerto, Ramos, & Rodríguez-Chía, 2011). The objective function of DOMP applies a penalty to the cost of supplying a client which is dependent on the *position* of that cost relative to the costs of supplying the remaining clients. Therefore, it increases the flexibility in the modeling phase through rank dependent compensation factors which allow to model which party is the driving force in a supply chain.

In the last years, a number of algorithms have been developed to attack the resolution of DOMP (see Boland et al., 2006; Kalcsics et al., 2010a, 2010b; Marín, Nickel, Puerto, & Velten, 2009; Nickel, 2001; Nickel & Puerto, 1999 & Rodríguez-Chía et al., Rodríguez-Chía, Nickel, Puerto, & Fernández, 2000). The first exact method was a branch and bound (B& B) algorithm presented in Boland et al. (2006). Later, a more specialized formulation was introduced in Marín et al. (2009) giving rise to a more efficient branch and cut (B&Cut) algorithm. Finally, in Marín, Nickel, and Velten (2010) the authors develop a new formulation that has allowed to solve larger size instances to optimality. The reader is referred to Marín et al. (2010) for a comprehensive literature review on exact methods for DOMP. More recently, the capacitated version of these

* Corresponding author.
E-mail addresses: puerto@us.es (J. Puerto), Dperez@ull.es (D. Pérez-Brito), Cggarcia@ull.es (C.G. García-González).

0377-2217/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2013.09.029

problems has been also considered in Kalcsics et al., 2010a, 2010b where first attempts to solve capacitated versions of DOMP have been developed. However, none of these approaches leads to satisfactory results concerning the solution times of even medium size instances. In spite of that, the literature on heuristic algorithms for this family of problems is rather reduced. Domínguez-Marín, Nickel, Hansen, and Mladenović (2005) present two heuristic approaches, a Variable Neighborhood Search (VNS) and a genetic algorithm, for solving DOMP, whereas Stanimirovic, Kratica, and Dugosija (2007) proposes two Evolutionary Programs (with two different encodings: binary in HGA1 and integer in HGA2) based on new encodings of the solution for better evaluation of the objective function that improve the heuristic algorithms in Domínguez-Marín et al. (2005). In both papers, the authors use ORLIB *N*-median instances with up to 900 nodes for testing their results.

The goal of this paper is to develop a modified VNS heuristic for DOMP which takes some advantage of the new available knowledge on the structure of this problem. Specifically, we use refined neighborhoods' structures that favor faster improvement of the objective function in the local search phase. Moreover, we apply an efficient encoding of solutions avoiding sorting in each objective function evaluation. This specific encoding allows to obtain a faster implementation than the one in Domínguez-Marín et al. (2005) for the VNS paradigm to this family of problems. In addition, it provides results that compare with the best heuristic algorithms known so far for the DOMP (Stanimirovic et al., 2007). To this end, the paper is organized as follows. Section 2 is devoted to recall the DOMP. Section 3 describes our modified VNS algorithm for the DOMP. There, we describe our neighborhoods' structures and the different elements that allow a better encoding of solutions avoiding sorting in each evaluation of the objective function. The presentation of the algorithm is modular. Thus, we present the different functions that are used in the algorithm, namely, *Initial Solution*, *Variable Neighborhood Descent*, *Shaking* and finally the actual algorithm *Modified Variable Neighborhood Search*. In Section 4, we report our computational results based on 8 problem types, previously considered in the literature, and on data taken from the benchmark instances of the ORLIB *N*-median library by Beasley (1990). The paper ends with some conclusions on the proposed algorithm and on the comparisons with previously available heuristics for the considered problem.

2. The discrete ordered median problem

In order to introduce the *Discrete Ordered Median Problem* (DOMP) formally, we define a set V of M discrete locations. These locations represent clients as well as potential plant locations.

Moreover, let $C = [c_{ij}]$ $(i, j = 1, \ldots, M)$ be a non–negative $M \times M$ cost matrix, whereas c_{ij} denotes the cost of satisfying the total demand of client i from a plant at location j. Thereby, we assume that $c_{ii} = 0$ $(\forall\ i = 1, \ldots, M)$. This property of C is called *free self–service* (FSS). For the sake of readability, we denote by G the number of different values assumed by the elements of matrix C. Moreover, we shall refer to the sorted values of C by $c_{(j)}$, $j = 1, \ldots, G$.

Let N with $1 \leqslant N \leqslant M-1$ be the number of new plants which have to be located at the candidate sites. Then, the costs for satisfying the demand of the respective clients, given a feasible solution $X \subset V$ with $|X| = N$, can be represented by the following vector

$$c(X) := (c_1(X), \ldots, c_M(X)) \quad \text{with } c_i(X) = \min_{j \in X}\{c_{ij}\}\ \forall\ i \in V.$$

However, due to the desired flexibility, $c(X)$ cannot directly be used to define the objective function of the DOMP. Instead, consider a permutation σ_X on $\{1, \ldots, M\}$ for which the inequalities

$$c_{\sigma_X(1)}(X) \leqslant c_{\sigma_X(2)}(X) \leqslant \cdots \leqslant c_{\sigma_X(M)}(X)$$

hold. Using this permutation we define the *sorted cost vector* $c_{\leqslant}(X)$ corresponding to a feasible solution X as follows:

$$c_{\leqslant}(X) := (c_{\sigma_X(1)}(X), \ldots, c_{\sigma_X(M)}(X))$$

or for short

$$c_{\leqslant}(X) := (c_{(1)}(X), \ldots, c_{(M)}(X)).$$

Furthermore, let $\lambda = (\lambda_1, \ldots, \lambda_M)$ be an M-dimensional vector, with $\lambda_i \geqslant 0$ $(\forall\ i = 1, \ldots, M)$ representing a weight on the ith lowest component of the cost vector $c(X)$. Using the notation explained above the DOMP is defined as:

$$\min_{\substack{X \subset V \\ |X| = N}} f_\lambda(X) = \sum_{i=1}^{M} c_{(i)}(X) \cdot \lambda_i. \tag{1}$$

The function $f_\lambda(X)$ is called *ordered median function*. An example illustrating the structure of the DOMP and the calculation of the ordered median function is given below.

Example 2.1. Let $V = \{1, \ldots, 5\}$ and assume that $N = 2$ plants have to be located. Moreover, let the cost matrix C be as follows:

$$C = \begin{pmatrix} 0 & 4 & 5 & 3 & 3 \\ 1 & 0 & 6 & 2 & 2 \\ 7 & 3 & 0 & 3 & 1 \\ 7 & 3 & 5 & 0 & 5 \\ 1 & 3 & 2 & 3 & 0 \end{pmatrix}.$$

Clearly $G = 8$ and $c_{(1)} = 0$, $c_{(2)} = 1$, $c_{(3)} = 2$, $c_{(4)} = 3$, $c_{(5)} = 4$, $c_{(6)} = 5$, $c_{(7)} = 6$, $c_{(8)} = 7$.

With $\lambda = (0,0,1,1,0)$, an optimal solution of this problem instance is $X = \{1,4\}$. Therefore, the demand of locations 1, 2 and 5 are satisfied by plant 1 whereas the demand of the remaining locations are satisfied by plant 4. Hence, $c(X) = (0,1,3,0,1)$, $c_{\leqslant}(X) = (0,0,1,1,3)$ and

$$f_\lambda(X) = 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 3 = 2.$$

Note that by using appropriate values for λ, nearly all classical discrete facility location problems can be modeled by the above definition. In addition, a wide range of new and interesting problems can be derived. Some of these modeling possibilities are given in Table 1. For a more extensive list the interested reader is referred to Domínguez-Marín (2003) and Nickel and Puerto (2005).

Since the DOMP contains, as a special instance, the discrete *N*-median problem which is $\mathcal{NP}$-hard (see Kariv & Hakimi, 1979) the DOMP is also $\mathcal{NP}$-hard. In spite of that, as mentioned in the Introduction, different integer linear programming formulations have also been proposed for DOMP which can solve to optimality medium size instances up to 100 nodes (Marín et al., 2009). Nevertheless, for larger sizes the exact approaches do not perform well. In the following section we propose a modified VNS heuristic for DOMP.

Table 1
Modeling possibilities.

λ	$f_\lambda(X)$	Meaning
$(1, \ldots, 1)$	$\sum_{i=1}^{M} c_i(X)$	*N*-Median
$(0, \ldots, 0, 1)$	$\max_{1 \leqslant i \leqslant M} c_i(X)$	*N*-Center
$(\alpha, \ldots, \alpha, 1)$ $\alpha \in [0, 1]$	$\alpha \cdot \sum_{i=1}^{M} c_i(X) + (1-\alpha) \cdot \max_{1 \leqslant i \leqslant M} c_i(X)$	α-Centdian
$(0, \ldots, 0, \underbrace{1, \ldots, 1}_{k})$	$\sum_{i=M-k+1}^{M} c_{(i)}(X)$	*k*–Centrum

3. A modified variable neighborhood search for the DOMP

The basic idea of VNS is to implement a systematic change of neighborhood within a local search algorithm (see Hansen & Mladenović (1997, 2001, 2001, 2003), Hansen, Mladenović, & Moreno-Pérez (2010) and Hansen, Mladenović, & Pérez-Brito (2001)). Exploration of these neighborhoods can be done in two ways. The first one consists of systematically exploring the small neighborhoods, i.e. those closest to the current solution, until a better solution is found. The second one consists of partially exploring the large neighborhoods, i.e., those far from the current solution, by drawing a solution at random from them and beginning a (variable neighborhood) local search from there. The algorithm remains in the same solution until a better solution is found and then jumps there. These algorithms rank the neighborhoods to be explored in such a way that they are increasingly far from the current solution. We may view VNS as a way of escaping local optima, i.e., a "shaking" process, where movement to a neighborhood further from the current solution corresponds to a harder shake. In contrast to random restart, VNS allows a controlled increase in the level of the shake.

The standard application of VNS to the N-median problem proposed by Hansen and Mladenović (1997) encodes a solution by the indices of open facilities. Then, it needs to identify the allocation of demand points to open facilities to get the vector of costs. For the N-median problem updating the value of the objective function can be done step by step. As mentioned above N-median is a particular case of DOMP. For this reason Domínguez-Marín et al. (2005) applied the above structure to develop the first VNS algorithm for the DOMP. However, computation of the objective function value is much harder for DOMP than for N-median. Indeed, a major difficulty in the application of the above encoding to DOMP is to compute the variation between the objective function values when an interchange between two facilities is performed. This is so because one is forced to update and sort the whole cost vector after this interchange takes place. As a consequence, the complexity of this procedure applied to DOMP is higher due to this extra sorting which has to be done at each objective function calculation.

Our approach is different. First of all, we refine an already used family of neighborhoods' structures in a way that they favor faster local improvements in the objective function. This is improvement is attained because they bound, from above, the cost c_{ij} of allocations that are permitted in the considered neighborhood. This helps to speed up the local search phase. Moreover, the encoding is different.

Let us denote by $S = \{(X, a)$: X is a set of N potential locations of the new facilities and a is an allocation function $a(i) \in X$ for all $i \in \{1,\ldots,M\}\}$ a solution space of the problem. The solutions in S admit different neighborhoods' structures depending on the number of new facilities $k = 1, \ldots, k_{max}$, $(k_{max} \leqslant N)$ from the current solution that are replaced in the new solution; and on the properties of the allocation function a that is applied. We consider allocation functions a_r, $r \in \{c_{(1)},\ldots,c_{(G)}\}$, such that for any set X'_k of new facilities to be included in the current solution X, satisfy $a_r(i) \neq j$ for all $j \in X'_k$ such that $c_{ij} > r$. (New allocations at a cost greater than r are forbidden.)

We denote by $\mathcal{N}_{kr}$, $k \in \{1,\ldots,k_{max}\}$, $r = 0, \ldots, r_{max}$ $(k_{max} \leqslant N$, $r_{max} \leqslant c_{(G)})$ the set of such neighborhoods' structures and by $\mathcal{N}_{kr}(X)$ the set of solutions defining the neighborhood $\mathcal{N}_{kr}$ of a current solution X. More formally

$$X_1 \in \mathcal{N}_{kr}(X_2) \Longleftrightarrow |X_1 \setminus X_2| = k \text{ and } a_r(i) \neq j \quad \forall j \in X_1 \setminus X_2 \text{ such that } c_{ij} > r. \tag{2}$$

Note that the cardinality of $\mathcal{N}_{kr}(X)$ is of the order of $O(N^k(M-N)^k M^N)$, since k out of N facilities are dropped, k out of $M - N$ added into the solution and different allocations can be done. Finally, we observe that the union of the sets $\mathcal{N}_{kr}(X)$ together with X, is S.

3.1. Encoding and Evaluating Solutions

We represent solutions by open facilities but instead of using directly the vector of cost allocations, we maintain a list, $\mathcal{L}$, with minimal required information to evaluate the objective function without sorting after each solution update. See (Mladenović, Labbé, & Hansen, 2003 and Stanimirovic et al., 2007) for related encodings valid for N-center and N-ordered median problems, respectively.

Associated with each column j of the cost matrix, we define a list $\mathcal{L}(j)$. This list contains as many entries as the number of different values that appear in the column j of the cost matrix and it is sorted in increasing order of these values. Each record in the list $\mathcal{L}(j)$ keeps information on the facility, 'F', whose allocation costs are given in column j, a realizable allocation cost, '$c_{(.)}$', of serving from facility at F and the list of pointers 'R' to the rows where each cost in the column j appear in the cost matrix. This information describes any instance of a DOMP. In addition, each record has two more operational fields used for evaluation. The first one, called *used*, will keep track of the number of times that a demand point is served with the corresponding cost in the current solution whereas the second one will maintain the partial evaluation of the objective function (i.e., the cumulated cost) f_λ. (See Table 2 for an illustrative example of lists $\mathcal{L}(j)$ in Example 3.1.) Note that we have M lists that correspond to the M columns of the matrix C.

The list $\mathcal{L}$ is computed once in the preprocessing phase and it is the result of a merge-sort, by the field $c_{(.)}$, of the sorted lists $\mathcal{L}(j)$, $j = 1, \ldots, M$. (Table 3 shows the list $\mathcal{L}$ in Example 3.1.)

From the list $\mathcal{L}$ we generate Table $T(\mathcal{L})$ which has M rows. (Table 4 shows the Table $T(\mathcal{L})$ of Example 3.1.) Row j has as many elements as $|\mathcal{L}(j)|$. The $i-th$ entry of this row refers to $\mathcal{L}_i(j)$, the $i-th$ element of the list $\mathcal{L}(j)$. It contains the pair (k,R_i) where k is the position of $\mathcal{L}_i(j)$ in $\mathcal{L}$ and R_i is the field R in $\mathcal{L}_i(j)$. (Recall that R_i is the list of rows in the $j-th$ column of C whose values, $c_{\cdot j}$, are equal to the field $c_{(.)}$ of $\mathcal{L}_i(j)$).

Now, we can easily evaluate any solution of DOMP. Indeed, consider the solution $X = \{F_{i1},\ldots,F_{iN}\}$. Let $L(X)$ be the list that results from the merge-sort of the rows $i1, \ldots, iN$ of Table $T(\mathcal{L})$

Table 2
The lists $\mathcal{L}(j)$, $j = 1, \ldots, 5$.

Column j	Lists $\mathcal{L}(j)$, $j = 1, \ldots, 5$				
1	$\begin{bmatrix} 1 \\ (1) \\ 0 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 1 \\ (2,5) \\ 1 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 1 \\ (3,4) \\ 7 \\ \bullet \\ \bullet \end{bmatrix}$		
2	$\begin{bmatrix} 2 \\ (2) \\ 0 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 2 \\ (3,4,5) \\ 3 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 2 \\ (1) \\ 4 \\ \bullet \\ \bullet \end{bmatrix}$		
3	$\begin{bmatrix} 3 \\ (3) \\ 0 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 3 \\ (5) \\ 2 \\ \bullet \\ \bullet \end{bmatrix}$	$F \rightarrow$ 3 $R \rightarrow$ (1,4) $c_{(.)} \rightarrow$ 5 *used* $\rightarrow$ • $f_\lambda \rightarrow$ •	$\begin{bmatrix} 3 \\ (2) \\ 6 \\ \bullet \\ \bullet \end{bmatrix}$	
4	$\begin{bmatrix} 4 \\ (4) \\ 0 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 4 \\ (2) \\ 2 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 4 \\ (1,3,5) \\ 3 \\ \bullet \\ \bullet \end{bmatrix}$		
5	$\begin{bmatrix} 5 \\ (5) \\ 0 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 5 \\ (3) \\ 1 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 5 \\ (2) \\ 2 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 5 \\ (1) \\ 3 \\ \bullet \\ \bullet \end{bmatrix}$	$\begin{bmatrix} 5 \\ (4) \\ 5 \\ \bullet \\ \bullet \end{bmatrix}$

Table 3
The main list $\mathcal{L}$.

List $\mathcal{L}$									
	1	2	3	4	5	6	7	8	9
F	1	2	3	4	5	1	5	3	4
R	(1)	(2)	(3)	(4)	(5)	(2,5)	(3)	(5)	(2)
$c_{(\cdot)}$	0	0	0	0	0	1	1	2	2
used	•	•	•	•	•	•	•	•	•
f_λ	•	•	•	•	•	•	•	•	•
	10	11	12	13	14	15	16	17	18
F	5	2	4	5	2	3	5	3	1
R	(2)	(3,4,5)	(1,3,5)	(1)	(1)	(1,4)	(4)	(2)	(3,4)
$c_{(\cdot)}$	2	3	3	3	4	5	5	6	7
used	•	•	•	•	•	•	•	•	•
f_λ	•	•	•	•	•	•	•	•	•

Table 4
The table $T(\mathcal{L})$.

1	(1, (1))	(6, (2, 5))	(18, (3, 4))		
2	(2, (2))	(11, (3, 4, 5))	(14, (1))		
3	(3, (3))	(8, (5))	(15, (1, 4))	(17, (2))	
4	(4, (4))	(9, (2))	(12, (1, 3, 5))		
5	(5, (5))	(7, (3))	(10, (2))	(13, (1))	(16, (4))

in increasing order of k and without repetitions of any element in the lists R of the previously chosen pairs (k,R). Thus, if the next element to be merged in a partially built list $L(X)$ is $(\bar{k},\bar{R})$ with $\bar{R}=\bar{R^1}\bigcup\bar{R^2}$ and the elements in $\bar{R^1}$ were already covered by previous elements inserted in $L(X)$, then we only add the pair $(\bar{k},\bar{R^2})$. (Expression (5) shows $L(\{2,4\})$ in Example 3.1.)

The next example illustrates the data structure used in our algorithm.

Example 3.1. Consider the cost matrix introduced in Example 2.1. First, we have computed the lists $\mathcal{L}(j)$, $j=1,\ldots,5$. (See Table 2.) For instance, the third element in the third row, namely $[3,(1,4),5,\bullet,\bullet]^t$, comes from the third column of the cost matrix C and is built in the following way:

- $F=3$. It indicates that this record (vector) refers to the third column of the cost matrix C.
- $R=(1,4)$. The allocation costs $c_{13}=c_{43}$ assume the value $c_{(\cdot)}=5$. This field of R informs that the cost to which this element refers to, namely 5 (the sixth component of the vector of sorted costs), appears twice in column 3 of C, namely in rows 1 and 4.
- $c_{(\cdot)}=5$. The cost matrix to which we refer to is 5.

From the lists $\mathcal{L}(j)$, it is easy to obtain the main list $\mathcal{L}$ by simply merging the lists $\mathcal{L}(j)$ in non-decreasing value of their fields $c_{(\cdot)}$. Applying this merging to the elements in Table 2 results in the list $\mathcal{L}$ in Table 3.

Once we have obtained the main list $\mathcal{L}$, we generate the Table $T(\mathcal{L})$ (see Table 4). Recall that each element is a pair (record number,field R). For instance (15,(1,4)), in the third row, means that the cost 5 that appears in the record number 15 of $\mathcal{L}$ comes from rows 1 and 4.

All the above is done only once in the preprocessing phase.

In order to evaluate the objective function we use the following recursion. Assume that $|L(X)|=\ell$. Thus, we will consider, $ET(L(X))$, the evaluation table of $L(X)$ that has ℓ columns. (See Table 6 in Example 3.1, $\ell=4$.) We identify the columns in that table by their consecutive indices. Then, we refer to the elements of each column by their names and the superscript of the column, namely F^j, $c^j_{(\cdot)}$, $used^j$, f^j_λ point to the corresponding fields of the jth column in the evaluation table. For instance, in Example 3.1, $j=2$ in Table 6 refers to the second column which in fact comes from the fourth element in $\mathcal{L}$. Then, $F^2=4$ and $c^2_{(\cdot)}=0$. Moreover, let us define $\overline{used}^j$, $j=0,\ldots,\ell$, as the accumulated *'used'* field up to the jth column of $ET(L(X))$. By convention we assume that $\overline{used}^0=0$. For instance, in Table 6 of Example 3.1, $\overline{used}^3=used^1+used^2+used^3=1+1+2=4$ and $\overline{used}^4=5$.

Now, we can properly state the formula of f^j_λ for $j=0,\ldots,\ell$. First of all,

$$f^0_\lambda=0, \tag{3}$$

$$f^j_\lambda=f^{j-1}_\lambda+\sum_{k=\overline{used}^{j-1}+1}^{\overline{used}^j}\lambda_k c^j_{(\cdot)}. \tag{4}$$

As mentioned above, from formula (4), we see that the evaluation of f_λ is linear, once the evaluation Table $L(X)$ is obtained. (No sorting is needed.)

Example 3.1 (*Continuation*). With the information previously obtained, we can easily obtain the evaluation of any solution, for instance $\{2,4\}$, for $\lambda=(0,0,1,1,0)$, namely $f_{(0,0,1,1,0)}(\{2,4\})$. First, we obtain $L(\{2,4\})$ by merging rows 2 and 4 of Table $T(\mathcal{L})$ up to an accumulated frequency of 5 different elements in the second field R. (The reader may observe that in each row of the above table, the cardinality of the union of the elements of the second field of the pairs in any row of $T(\mathcal{L})$ is always M.)

$$L(\{2,4\})=[(2,(2)),(4,(4)),(11,(3,5)),(12,(1))]. \tag{5}$$

Table 6 shows $ET(L(\{2,4\}))$. Bold columns in Table 5 induce the evaluation table in Table 6. The objective value of the solution $\{2,4\}$ is given by the last field f_λ of the last record in the evaluation table $ET(L(\{2,4\}))$. In this case the last element of $L(\{2,4\})$ is (12,(1)) which means that we have to consider the record number 12 in the list $\mathcal{L}$ but only for client 1 (see Table 6).

Finally, to compute the value of the objective function $f_{(0,0,1,1,0)}(\{2,4\})$ one simply needs to process, using the recursive formula (4), the four records shown in Table 6. The value of the objective function for the solution $\{2,4\}$ is 6.

Note that changing the solution from $\{2,4\}$ to $\{1,4\}$ can be done by merging N rows of $T(\mathcal{L})$. First, we compute the list $L(\{1,4\})$ by removing from $L(\{2,4\})$ the elements that correspond to the facility F_2 (row number 2 of $T(\mathcal{L})$) and then we insert the elements from row number 1 in $T(\mathcal{L})$, up to a frequency of 5 (using that repetitions of elements in R are not allowed). The resulting list is:

$$L(\{1,4\})=[(1,(1)),(4,(4)),(6,(2,5)),(12,(3))].$$

Again, the evaluation of the objective function $f_{(0,0,1,1,0)}(\{1,4\})$ simply needs to process the following four records shown in Table 7.

With the recursive formula (4), the value of the objective function for the solution $\{1,4\}$ is 2.

Table 5
The main list $\mathcal{L}$. Bold columns are associated to the solution defined by facilities {2,4}.

List $\mathcal{L}$									
	1	**2**	3	**4**	5	6	7	8	9
F	1	**2**	3	**4**	5	1	5	3	4
R	(1)	(**2**)	(3)	(**4**)	(5)	(2,5)	(3)	(5)	(2)
$c_{(\cdot)}$	0	**0**	0	**0**	0	**1**	1	2	2
used	•	**1**	•	**1**	•	•	•	•	•
f_λ	•	**0**	•	**0**	•	•	•	•	•
	10	**11**	**12**	13	14	15	16	17	18
F	5	**2**	**4**	5	2	3	5	3	1
freq	(2)	(**3**,**4**,**5**)	(**1**,**3**,**5**)	(1)	(1)	(1,3)	(4)	(2)	(3,4)
R	2	**3**	**3**	3	4	5	5	6	7
$c_{(\cdot)}$	•	**2**	**1**	•	•	•	•	•	•
used	•	**6**	**6**	•	•	•	•	•	•
f_λ									

Table 6
Evaluation Table $ET(L(\{2,4\}))$ and objective function evaluation.

	2	4	11	12
F	2	4	2	4
R	(2)	(4)	(3,4,5)	(1,3,5)
$c_{(\cdot)}$	0	0	3	3
used	1	1	2	1
f_λ	$0\times 0=0$	$0+0\times 0=0$	$0+1\times 3+1\times 3=6$	$6+0\times 3=6$

Table 7
Evaluation Table $ET(L(\{1,4\}))$ and objective function evaluation.

	1	4	6	12
F	1	4	1	4
R	(1)	(4)	(2,5)	(1,3,5)
$c_{(\cdot)}$	0	0	1	3
used	1	1	2	1
f_λ	$0\times 0=0$	$0+0\times 0=0$	$0+1\times 1+1\times 1=2$	$2+0\times 3=2$

In the following, we describe the modified version of the VNS algorithm for the DOMP. In the presentation we follow a modular description. We will describe, in subsections, the different subroutines that we use in the main algorithm. Then, we present its pseudocode. In the description of the heuristic, we use the following notation:

- x_{cur}: current solution (new facilities);
- f_{cur}: current incumbent objective function value;
- *goin*: index set of the facilities to be inserted in the current solution;
- *goout*: index set of the facilities to be deleted from the current solution;
- g^*: current objective function value obtained in local procedures.

In the following four subsections we describe the components of our modified VNS heuristic for DOMP.

3.2. Initial solution

In this subsection we present the approach that we have followed to construct a solution to initialize our VNS heuristic. In order to choose the initial solution, we have compared several strategies on medium size instances of the problem (pmed16–pmed25) to conclude which one should be considered in the overall computational study. We have tested the following methods:

1. Pure deterministic. Taking as initial solutions those facilities numbered $\{1,\ldots,N\}$ in the entire list of facilities.
2. Pure randomization. Choosing N facilities at random.
3. A Greedy algorithm. It works as follows: The first chosen facility is the one that minimizes the ordered median objective function assuming that we are interested in the 1-facility case. After that, in every step we choose the facility with minimal objective function value taking into account the facilities already selected. This procedure terminates as soon as N facilities are chosen.
4. A Random-Greedy construction (RG-construction) (see e.g. Resende & González-Velarde, 2003). Our construction works by initially choosing $\lfloor N/2 \rfloor$ facilities at random and then completing them until N, applying the greedy approach described above.

We have tested the above methods in order to find the one with the best performance for our algorithm. Some of them result in very poor performance (as for instance the pure deterministic) and therefore were discarded. The remaining methods were combined in order to have the best possible initial solution in any of the ten executions that we run our heuristic for each instance. In order to do that, we tested several initialization strategies on medium size instances (pmed16–pmed25) for all problem types and a fixed upper limit of 300 seconds of CPU time. We report in Table 8 the performance of the three best strategies found: 10 executions with greedy initial solution (10 G), 1 execution greedy and 9 random (1G-9R) and 10 executions with initial solution from the RG-construction (10 RG).

The best performance was obtained by the RG-construction for all instances and problem types and therefore, we decided to apply it as the initialization procedure within our heuristic (see Table 8).

Table 8
Gap for different initialization schemes on problem types T1–T8 and fixed upper limit of 300 seconds of CPU time.

Problem	T1 (%)			T2 (%)			T3 (%)		
	10 G	1G-9R	10-GR	10 G	1G-9R	10-GR	10 G	1G-9R	10-GR
pmed16	0.02	0.00	0.00	6.38	0.00	0.00	0.00	0.00	0.00
pmed17	0.01	0.00	0.00	2.56	2.56	0.00	0.00	0.00	0.00
pmed18	0.08	0.00	0.00	32.14	32.14	32.14	0.03	0.00	0.00
pmed19	0.67	0.04	0.00	61.11	61.11	22.22	0.22	0.00	0.00
pmed20	0.28	0.28	0.22	130.77	130.77	61.54	0.08	0.08	0.08
pmed21	0.02	0.00	0.00	7.50	0.00	0.00	0.07	0.00	0.00
pmed22	1.05	0.00	0.00	2.63	2.63	2.63	0.09	0.00	0.00
pmed23	0.09	0.00	0.00	36.36	36.36	36.36	0.04	0.04	0.04
pmed24	0.20	0.00	0.00	53.33	53.33	53.33	0.16	0.16	0.16
pmed25	0.71	0.05	0.05	100.00	100.00	63.64	0.23	0.23	0.00
Avrg. GAP	0.29	0.04	0.03	43.28	41.89	27.19	0.09	0.05	0.03
	T4			T5			T6		
	10 G	1G-9R	10-GR	10 G	1G-9R	10-GR	10 G	1G-9R	10-GR
pmed16	0.00	0.00	0.00	0.05	0.00	0.00	0.05	0.00	0.00
pmed17	0.11	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00
pmed18	0.12	0.00	0.00	0.08	0.00	0.00	0.13	0.00	0.00
pmed19	0.31	0.00	0.00	0.35	0.00	0.00	0.35	0.14	0.14
pmed20	0.70	0.35	0.35	0.44	0.33	0.33	0.67	0.56	0.00
pmed21	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pmed22	0.04	0.00	0.00	0.07	0.00	0.00	0.07	0.00	0.00
pmed23	0.18	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00
pmed24	0.20	0.10	0.10	0.54	0.00	0.00	0.27	0.00	0.00
pmed25	0.35	0.17	0.00	0.65	0.22	0.11	0.44	0.00	0.00
Avrg. GAP	0.21	0.06	0.04	0.25	0.05	0.04	0.22	0.07	0.01
	T7			T8					
	10 G	1G-9R	10-GR	10 G	1G-9R	10-GR			
pmed16	0.00	0.00	0.00	0.00	0.00	0.00			
pmed17	0.09	0.00	0.00	0.13	0.00	0.00			
pmed18	0.12	0.09	0.06	0.37	0.37	0.12			
pmed19	0.16	0.00	0.00	0.42	0.00	0.00			
pmed20	0.34	0.00	0.00	0.50	0.50	0.50			
pmed21	0.00	0.00	0.00	0.00	0.00	0.00			
pmed22	0.03	0.00	0.00	0.00	0.00	0.00			
pmed23	0.10	0.00	0.00	0.51	0.00	0.00			
pmed24	0.20	0.00	0.00	0.71	0.00	0.00			
pmed25	0.41	0.16	0.16	0.16	0.16	0.00			
Avrg. GAP	0.14	0.03	0.02	0.28	0.10	0.06			

3.3. Local search: Variable Neighborhood Descent (VND)

This procedure shows how to modify/update the objective function when a given solution is updated. In our descent phase we search deterministically within the neighborhoods' structures given by the family N_{kr} described above. We use only two different k-interchange neighborhoods' structures N_{kr} corresponding to $r \in \{\lceil c_{(G)}/2 \rceil, c_{(G)}\}$. This means that when searching within the first neighborhood structure $(N_{k\lceil c_{(G)}/2 \rceil})$, we only allow allocation costs that are less than or equal to $\lceil c_{(G)}/2 \rceil$, whereas in the second neighborhood structure $(N_{kc_{(G)}})$ we allow any allocation cost. The maximal cardinality k'_{max} of the set of facilities to be added to a partial solution in our VND is a parameter that must be chosen in the implementation. Note that this parameter k'_{max} determines the neighborhood structure N_{kr} used in the search phase. First of all, we have used two approaches to choose the set, *goin*, of new facilities to be added to the current solution. The first one is a *First improvement* scheme that looks, in the neighborhood of the current solution, only for the first improvement found. The second one is a *Best improvement* that looks for new solutions amongst all the neighbors of the current solution in order to choose the best one. These two approaches to add facilities to the current solution have been combined with the different strategies, used in the routine *goout*, to select the facilities to be removed from the current solution. (The different combinations can be seen in Table 9.)

Since, we have already described the two methods for choosing the set *goin*, we will assume, from now on, that the set of facilities that is added to the current solution is known.

We have tested two methods for choosing the set *goout*, of facilities to be removed from a solution. The first one does a non-guided search on the current neighborhood within a pre-specified time limit and the second one selects these facilities according to a list of preferences $\mathcal{P}$, that provides a heuristic order to choose solutions within the neighborhoods' structures. This list is computed only once at the preprocessing phase.

Table 9
Local search routines.

IN–OUT	Tmax = 30 seconds.			
	First-Guided	Best-Guided	First-Best	Best–Best
T1				
Avrg. Obj.	6273.40	6178.33	6494.65	6318.85
Avrg. Gap (%)	23.95	22.30	29.63	27.67
Num. Opt.	1	14	0	22
T4				
Avrg. Obj.	4696.80	4624.03	4905.33	4756.08
Avrg. Gap (%)	24.33	23.04	32.22	30.99
Num. Opt.	2	16	0	21

The rationale of this heuristic order is based on the following construction. For each feasible facility j we compute the value $f_\lambda(\{j\})$, i.e., the solution of the 1-ordered median problem provided that the only open facility is j. The greater the value f_λ, the lower the performance that one expects for that facility in any solution. Therefore, we set the list of preferences in decreasing order with respect to these values. Algorithms 3.1 and 3.2 describe the two methods to choose the set *goout*, namely by the *Best update* (non-guided) or by the guided list of preferences, *Guided update*, respectively.

Algorithm 3.1. Best update ($c, \lambda, x_{cur}, f_{cur}, goin, M, N, k, \mathcal{P}, t_{max}$, var g^*, var $goout^*$)

Initialization;
Set $g^* \leftarrow \infty$;
Set $x_{ini} \leftarrow x_{cur}$ /* $|x_{ini}| = M$ */ ;
/* *Best deletion* */;
repeat
  Choose $goout \subset x_{ini}$ such that $|goout| = k < M$;
  Set $x_{cur} \leftarrow (x_{cur} \setminus goout) \cup goin$ /* $goin \subset (N \setminus M)$ */;
  Compute $f_\lambda(x_{cur})$ /* using the main list $\mathcal{L}$ and the Table $T(\mathcal{L})$ */;
  if $f_\lambda(x_{cur}) < g^*$ **then**
    $g^* \leftarrow f_\lambda(x_{cur})$, $goout^* \leftarrow goout$;
  end
until *(CPU() > t_{max}) or (Neighborhood explored)* ;

Finally, based on the different schemes to choose the sets *goin* and *goout*, the pseudocode of the local search VND routine that we have implemented within our VNS heuristic is described in the Algorithm 3.3.

In the pseudocode of Algorithm 3.3, we show a new version of VND, where *goin* is actually a set, i.e., in an exhaustive search it would enumerate all possible subsets of facilities with cardinality k. Nevertheless, in our implementation we use the strategy of stopping the exploration based on the number of unsuccessful attempts at improvement. Specifically, we terminate the exploration of this neighborhood if this value is greater than N.

Algorithm 3.2. Guided update ($c, \lambda, x_{cur}, f_{cur}, goin, M, N, k, \mathcal{P}, t_{max}$, var g^*, var $goout^*$)

Initialization;
Set $g^* \leftarrow \infty$;
Set $x_{ini} \leftarrow x_{cur}$ /* $|x_{ini}| = M$ */ ;
/* *Guided deletion* */ ;
repeat
  Choose $goout \subset x_{ini}$ with $|goout| = k$ and in circular sequence within $\mathcal{P}$;
  Set $x_{cur} \leftarrow (x_{cur} \setminus goout) \cup goin$ /* $goin \subset (N \setminus M)$ */;
  Compute $f_\lambda(x_{cur})$ /* using the main list $\mathcal{L}$ and the Table $T(\mathcal{L})$ */;
  if $f_\lambda(x_{cur}) < g^*$ **then**
    $g^* \leftarrow f_\lambda(x_{cur})$, $goout^* \leftarrow goout$;
    exit
  end
until *(CPU() > t_{max}) or (Neighborhood explored)* ;

Table 10
Determining the size k_{max}.

	(Tmax = 600 seconds)					
	k_{max} = 3		k_{max} = 5		k_{max} = 10	
VNS with	Best-Guided	Best–Best	Best-Guided	Best–Best	Best-Guided	Best–Best
T1						
Avrg.Obj.	5750.18	6096.25	5766.18	6095.83	5799.73	6099.76
Avrg.Gap	7.67%	20.05%	6.24%	20.05%	9.05%	18.09
Num.Opt.	20	22	22	16	18	15
T4						
Avrg.Obj.	4283.53	4587.18	4291.48	4589.98	4307.05	4508.02
Avrg.Gap	6.13%	22.33%	5.43%	22.48%	7.27%	20.07
Num.Opt.	20	24	23	18	18	16

Algorithm 3.3. *VND* $(c, c_{(G)}, \lambda, x_{cur}, f_{cur}, goin, M, N, k'_{max}, \mathcal{P}$, var f', var $x'_{cur})$

Initialization;
Set $g^* \leftarrow f_{cur}$;
Iteration step ;
/* For each k it explores first $r = \lceil c_{(G)}/2 \rceil$ and if no improvement is found it moves to $r = c_{(G)}$ */ ;
for $k = 1 \dots k'_{max}$ **do**
  set $i \leftarrow 1$;
  while $i \leq 2$ **do**
    if *i=1* **then**
      $r \leftarrow \lceil c_{(G)}/2 \rceil$
    else
      $r \leftarrow c_{(G)}$
    end
    /* Add facilities *goin* to the current solution and choose *goout* to be removed according to *"Type"* of Update: Alg. 3.1 → *Best* or Alg. 3.2 → *Guided* */ ;
    $x \leftarrow \arg \min_{y \in N_{kr}(x_{cur})} f(x_{cur})$ /* Find the best neighbor in $N_{kr}(x_{cur})$ */;
    for each $goin \subset x \cap (N \setminus M)$ such that $|goin| = k$ **do**
      "Type" of Update $(c, \lambda, x_{cur}, g^*, goin, M, N, k, \mathcal{P}, g, goout)$;
      if $g < g^*$ **then**
        $g^* \leftarrow g$, $goin^* \leftarrow goin$, $goout^* \leftarrow goout$;
        $x^*_{cur} \leftarrow (x_{cur} \setminus goout^*) \cup goin^*$
      end
      if *Maximum running time exceeded* **then** *exit*
    end
    $i \leftarrow i + 1$;
  end
end
Termination /* Variables returned by calls to function *VND* */;
$f'_{cur} \leftarrow g^*$, $x'_{cur} \leftarrow x^*_{cur}$;

we conclude that the best combinations for the implementation of *goin–goout* are *Best-Guided* and *Best-Best* (see Table 9).

Table 9 reports the computational results of the different combinations of strategies for choosing the sets *goin* and *goout* in the VND routine. We have tested the approaches *First*/*Best* for *goin* and *Guided*/*Best* for *goout*, for the 40 benchmark instances considered (ORLIB *N*-median instances) and on problem types *T*1 and *T*4 (see Section 4 for their descriptions). The choices, *First*/*Best*, mean that selection for *goin* is based on first improvement/exhaustive search, respectively. On the other hand, the options *Guided*/*Best*, mean that selection for *goout* is based on the preference list described in Section 3.3 (see Algorithm 3.2) or on the best improvement (see Algorithm 3.1).

Table 9 has two blocks. The first one reports on the results for the problem *T*1 and the second one for problem *T*4 (see Section 4). We have run the VND routine with a time limit of 30 seconds for the four possible combinations of IN-OUT. Each block reports the average objective value, *Avrg. Obj.*; the average gap, *Avrg.Gap*, and the number of optimal solutions found, *Num.Opt.* for the 40 considered instances of *N-median* problems in ORLIB. From these results,

Table 11
CPU2000 results published by standards performance evaluation corporation.

CPU	INTEL P.IV 1.8 gigahertz	AMD Sempron 1.6 gigahertz	INTEL P.III 0.8 gigahertz
CPU speed	633	587	417
Factor	633/633	587/633	417/633

3.4. Shaking

In our implementation of the VNS, we have tested two shaking operators. The first one consists of choosing at random the set *goin* and selecting, according to a priority list, the set *goout*; whereas the second one chooses at random, both the sets *goin* and *goout*. Our computational experiments have shown that the second one performs better in terms of running times and quality of solutions. Therefore, our results are based on this second option.

In the *Shaking operator* step, the incumbent solution x_{opt} is perturbed in such a way that $|x_{cur} \setminus x_{opt}| = k$. Nevertheless, this step does not guarantee that x_{cur} belongs to $\mathcal{N}_{kr}(x_{opt})$ due to randomization of the choice of *goin* and possible reinsertion of the same facility after it has been removed in the *goout* phase. Then, x_{cur} is used as initial solution for VND routine in *Local Search* step. If a solution better than x_{opt} is obtained, we move there and start again with small perturbations of this new best solution, i.e., $k \leftarrow 1$. Otherwise, we increase the distance between x_{opt} and the new randomly generated point, i.e., we set $k \leftarrow k + 1$. If k reaches k_{max} (this parameter can be chosen equal to N), we return to *Main* step (within the modified VNS algorithm described in Section 3.5), i.e., the main step can be iterated until some other stopping condition is met (e.g. maximum number of iterations, maximum CPU time allowed, or maximum number of iterations between two improvements). Note that the element x_{cur} is generated at random in *Shaking operator* step in order to avoid cycling which might occur if any deterministic rule were used.

Table 10 reports the results of the computational tests that we have compared in order to set the maximum size of the family of neighbors to be used in the algorithm. We have compared three different sizes $k_{max} = 3, 5, 10$ on problem types $T1$ and $T4$ (see Section 4). In order to do that, we ran the algorithm with a time limit of 600 seconds and for the best two VND searches, namely "Best-Guided" and "Best–Best". From the results we conclude that the best results are obtained using "Best-Guided" as VND routine and with a number of facilities replaced in each iteration of $k_{max} = 5$ (see Table 10).

Algorithm 3.4. Modified VNS for the DOMP

Preprocessing;
Compute $\mathcal{L}$ the sorted list of records described in Section 3.1;
Compute the Table $T(\mathcal{L})$ with M rows whose elements are pairs of *(record number in $\mathcal{L}$, field R of the corresponding record in $\mathcal{L}$)*. (See Section 3.1);
Initialization;
Use RG-construction to generate x_{cur}, and f_{cur} as initialization of *Variable Neighborhood Descent*;
Set the neighborhoods' structures $\mathcal{N}_{kr}$, $k = 1, \ldots, k_{max}$, $r \in \{\lceil c_{(G)}/2 \rceil, c_{(G)}\}$. (See (2));
Choose stopping criterion ;
Main step;
$x'_{cur} \leftarrow x_{cur}$;
repeat
  $k \leftarrow 1$;
  repeat
    Shaking operator /* Gen. a sol. at random from the neighborhood, $\mathcal{N}_{k\lceil c_{(G)}/2 \rceil}$ */ ;
    Set *goin* choosing at random k facilities that are not in x'_{cur} ;
    Set *goout* choosing at random k facilities to be removed in x'_{cur} ;
    Update $x'_{cur} \leftarrow (x'_{cur} \setminus goout) \cup goin$;
    $f'_{cur} \leftarrow f_\lambda(x'_{cur})$ /* using the main list $\mathcal{L}$ and the Table $T(\mathcal{L})$ */ ;
    Local Search (VND) /* Call Algorithm 3.3: x'_{cur}, f'_{cur} returned values */;
    $VND(c, c_{(G)}, \lambda, x_{cur}, f_{cur}, goin, M, N, k'_{max}, \mathcal{P}, f'_{cur}, x'_{cur})$;
    Move or not Move;
    if $f'_{cur} < f_{cur}$ **then**
      /* Save current solution to be incumbent; return to $\mathcal{N}_{1\lceil c_{(G)}/2 \rceil}$ */ ;
      $f_{cur} \leftarrow f'_{cur}$; $x_{cur} \leftarrow x'_{cur}$;
      set $k \leftarrow 1$;
    else
      /* Current solution is the incumbent; change neighborhood */ ;
      $f'_{cur} \leftarrow f_{cur}$; $x'_{cur} \leftarrow x_{cur}$;
      set $k \leftarrow k + 1$;
    end
  until $(k = k_{max})$;
  $time \leftarrow CPU()$;
until $(time > t_{max})$ *or (stopping criterion is met)* ;
Termination;
Best solution and objective value found: x_{cur} and f_{cur}.

3.5. Modified variable neighborhood search

We present in Algorithm 3.4, the Modified Variable Neighborhood Search (MOD-VNS) heuristic for DOMP. In its description we refer to the procedures described in Section 3.4. The algorithm has three main phases: (1) *Preprocessing* that sets the data structure used in the evaluation and update of new solutions and (2) *Initialization* that sets the initial solution, stopping criterion, etc. Finally, the third phase *Main step* that is the main loop including the *Shaking operator* and the *Local Search (VND)* which actually runs the VNS heuristic.

4. Computational results

In this section, we compare the results of MOD-VNS algorithm with existing algorithms from the literature. All test were implemented in C and compiled with Microsoft Visual C++ 6.0 and were run on a Pentium 4 at 1.8 gigahertz. with 1 GB of RAM. Our results are based on the benchmark instances of the ORLIB N-median data set publicly available electronically from http://people.brunel.ac.uk/~mastjjb/jeb/info.html (see Beasley (1990)). As in previous papers on this topic, we considered N-median instances with $100 \leqslant M \leqslant 900$ nodes and $5 \leqslant N \leqslant 200$ potential new facilities and solved them for 8 different classes of λ-parameters already suggested in the literature. Each λ is associated with a different

Table 12
Results for **T1** (*N*-median).

Instance	M	N	*OPT*	HGA1			VNS			MOD-VNS		
				OBJ	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)
pmed1	100	5	5819	5819	1.64	0.00	5819	1.19	0.00	5819	0.00	0.00
pmed2	100	10	4093	4093	2.06	0.00	4093	2.97	0.00	4093	0.28	0.00
pmed3	100	10	4250	4250	1.89	0.00	4250	3.00	0.00	4250	0.00	0.00
pmed4	100	20	3034	3034	2.85	0.00	3046	5.98	0.40	3034	1.04	0.00
pmed5	100	33	1355	1355	3.35	0.00	1358	6.81	0.22	1355	0.87	0.00
pmed6	200	5	7824	7824	4.18	0.00	7824	7.95	0.00	7824	0.00	0.00
pmed7	200	10	5631	5631	5.84	0.00	5639	12.72	0.14	5631	0.00	0.00
pmed8	200	20	4445	4445	7.53	0.00	4457	21.05	0.27	4445	0.00	0.00
pmed9	200	40	2734	2734	8.15	0.00	2753	41.98	0.69	2734	31.26	0.00
pmed10	200	67	1255	1259	11.05	0.32	1259	72.22	0.32	1255	149.64	0.00
pmed11	300	5	7696	7696	7.30	0.00	7696	12.52	0.00	7696	0.00	0.00
pmed12	300	10	6634	6634	12.92	0.00	6634	26.02	0.00	6634	0.00	0.00
pmed13	300	30	4374	4374	13.26	0.00	4374	87.92	0.00	4374	0.00	0.00
pmed14	300	60	2968	2969	20.88	0.03	2969	241.95	0.03	2968	52.97	0.00
pmed15	300	100	1729	1736	26.85	0.40	1739	363.39	0.58	1729	23.00	0.00
pmed16	400	5	8162	8162	13.65	0.00	8162	24.36	0.00	8162	0.00	0.00
pmed17	400	10	6999	6999	24.81	0.00	6999	47.30	0.00	6999	2.02	0.00
pmed18	400	40	4809	4809	22.91	0.00	4811	275.69	0.04	4809	71.95	0.00
pmed19	400	80	2845	2851	46.71	0.21	2864	469.30	0.67	2845	286.63	0.00
pmed20	400	133	1789	1794	56.24	0.28	1790	915.17	0.06	1794	302.24	0.28
pmed21	500	5	9138	9138	15.32	0.00	9138	27.39	0.00	9138	0.00	0.00
pmed22	500	10	8579	8579	36.01	0.00	8669	64.25	1.05	8579	0.00	0.00
pmed23	500	50	4619	4624	41.36	0.11	4619	443.23	0.00	4619	18.63	0.00
pmed24	500	100	2961	2966	89.33	0.17	2967	1382.84	0.20	2961	333.94	0.00
pmed25	500	167	1828	1838	125.68	0.55	1841	2297.25	0.71	1829	225.66	0.05
pmed26	600	5	9917	9917	25.02	0.00	9917	48.45	0.00	9917	0.00	0.00
pmed27	600	10	8307	8330	39.92	0.28	8310	127.63	0.04	8307	0.00	0.00
pmed28	600	60	4498	4500	75.29	0.04	4508	965.48	0.22	4499	270.64	0.02
pmed29	600	120	3033	3036	149.39	0.10	3036	2758.56	0.10	3038	160.11	0.16
pmed30	600	200	1989	2008	194.85	0.96	2009	3002.34	1.01	1993	203.25	0.20
pmed31	700	5	10,086	10,086	25.86	0.00	10,086	56.02	0.00	10,086	0.00	0.00
pmed32	700	10	9297	9297	59.80	0.00	9301	165.27	0.04	9297	0.00	0.00
pmed33	700	70	4700	4719	105.54	0.40	4705	2311.03	0.11	4702	110.18	0.04
pmed34	700	140	3013	3027	237.58	0.46	3024	5384.19	0.37	3020	239.11	0.23
pmed35	800	5	10,400	10,400	34.14	0.00	10,400	88.50	0.00	10,400	0.00	0.00
pmed36	800	10	9934	9951	63.58	0.17	9934	200.97	0.00	9934	17.22	0.00
pmed37	800	80	5057	5063	124.30	0.12	5066	2830.30	0.18	5063	125.30	0.12
pmed38	900	5	11,060	11,060	47.65	0.00	11,060	150.53	0.00	11,060	0.00	0.00
pmed39	900	10	9423	9423	68.02	0.00	9423	200.73	0.00	9423	0.00	0.00
pmed40	900	90	5128	5133	374.80	0.10	5141	4774.38	0.25	5140	301.00	0.23
Avrg.			5535.30	5539.08	51.79	0.12	5542.25	493.63	0.19	5536.38	73.17	0.03
# Best				23	14		17	0		31	26	

objective function and it models different types of DOMP, as already indicated in Section 2. The λ-parameters considered in our experiments are:

- T1: $\lambda = (1,\ldots,1)$, vector corresponding to the *N*-median problem.
- T2: $\lambda = (0,\ldots,0,1)$, vector corresponding to the *N*-center problem.
- T3: $\lambda = (0,\ldots,0,\underbrace{1,\ldots,1}_{k})$, vector corresponding to the *k*-centrum problem, where $k = \lfloor \frac{M}{3} \rfloor$.
- T4: $\lambda = (\underbrace{0,\ldots,0}_{k_1},1,\ldots,1,\underbrace{0,\ldots,0}_{k_2})$, vector corresponding to the (k_1,k_2)-trimmed mean problem, where $k_1 = N + \lceil \frac{M}{10} \rceil$ and $k_2 = \lceil \frac{M}{10} \rceil$.
- T5: $\lambda = \begin{cases} (0,1,0,1,\ldots,0,1,0,1) & \text{if M is even} \\ (0,1,0,1,\ldots,0,1,0) & \text{otherwise} \end{cases}$, vector corresponding to an alternate series of zeroes and ones starting with zero.
- T6: $\lambda = \begin{cases} (1,0,1,0,\ldots,1,0,1,0) & \text{if M is even} \\ (1,0,1,0,\ldots,1,0,1) & \text{otherwise} \end{cases}$, vector corresponding to an alternate series of zeroes and ones starting with one.
- T7: $\lambda = \begin{cases} (0,1,1,0,1,1,\ldots,0,1,1,0,1,1) & \text{if } M \equiv 0 \pmod 3 \\ (0,1,1,0,1,1,\ldots,0,1,1,0) & \text{if } M \equiv 1 \pmod 3 \\ (0,1,1,0,1,1,\ldots,0,1,1,0,1) & \text{if } M \equiv 2 \pmod 3 \end{cases}$, vector corresponding to an alternate series of zeroes and two consecutive ones starting with zero.
- T8: $\lambda = \begin{cases} (0,0,1,0,0,1,\ldots,0,0,1,0,0,1) & \text{if } M \equiv 0 \pmod 3 \\ (0,0,1,0,0,1,\ldots,0,0,1,0) & \text{if } M \equiv 1 \pmod 3 \\ (0,0,1,0,0,1,\ldots,0,0,1,0,0) & \text{if } M \equiv 2 \pmod 3 \end{cases}$, vector corresponding to an alternate series of two consecutive zeroes and one starting with zero.

Our stopping criterion is based on running time. We fixed a time limit of 200 seconds for the small size instances (those up to 400 nodes) and we raised the time limit to 600 seconds for the large size instances (those with more than 400 nodes). We run MOD-VNS algorithm 10 times for each instance. In all the executions MOD-VNS is initialized by the solution of the RG-construction as described in Section 3.2. We report the best

Table 13
Results for **T2** (*N*-center) and **T3**.

T2							T3						
	HGA1			MOD-VNS				HGA1			MOD-VNS		
BEST	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)	*BEST*	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)
127	127	2.18	0.00	127	0.08	0.00	3148	3148	1.74	0.00	3148	0.70	0.00
98	98	2.50	0.00	98	1.59	0.00	2444	2444	2.36	0.00	2444	0.22	0.00
93	94	2.76	1.08	93	0.25	0.00	2452	2452	2.06	0.00	2452	0.19	0.00
74	79	3.72	6.76	74	14.29	0.00	1941	1961	2.32	1.03	1941	3.02	0.00
48	49	6.18	2.08	48	5.99	0.00	1072	1072	2.36	0.00	1072	1.34	0.00
84	84	6.10	0.00	84	0.69	0.00	4163	4163	4.00	0.00	4163	3.23	0.00
64	64	10.10	0.00	64	1.03	0.00	3157	3157	9.91	0.00	3157	2.53	0.00
55	58	15.05	5.45	55	91.94	0.00	2630	2630	6.46	0.00	2630	3.50	0.00
37	44	22.83	18.92	40	111.14	8.11	1844	1856	9.12	0.65	1844	77.40	0.00
20	32	20.63	60.00	31	82.49	55.00	931	933	11.21	0.21	931	2.18	0.00
59	59	9.68	0.00	59	14.52	0.00	4115	4115	7.83	0.00	4115	4.62	0.00
51	52	17.79	1.96	51	13.92	0.00	3613	3613	12.67	0.00	3613	13.56	0.00
36	41	40.78	13.89	40	134.25	11.11	2698	2705	12.79	0.26	2698	86.06	0.00
26	35	59.80	34.62	37	143.33	42.31	1935	1943	21.71	0.41	1935	15.01	0.00
18	26	65.55	44.44	29	116.53	61.11	1285	1293	27.22	0.62	1285	205.70	0.00
47	47	16.14	0.00	47	0.67	0.00	4220	4220	11.85	0.00	4220	9.88	0.00
39	40	35.99	2.56	39	38.73	0.00	3746	3746	23.38	0.00	3746	12.54	0.00
28	37	83.99	32.14	37	265.84	32.14	2859	2867	26.70	0.28	2859	25.16	0.00
18	29	114.32	61.11	22	109.85	22.22	1826	1826	30.29	0.00	1826	29.94	0.00
13	32	68.19	146.15	21	103.50	61.54	1311	1311	72.85	0.00	1312	286.10	0.08
40	40	21.83	0.00	40	1.13	0.00	4612	4612	20.24	0.00	4612	13.56	0.00
38	39	47.49	2.63	39	113.16	2.63	4466	4466	35.81	0.00	4466	18.03	0.00
22	30	135.25	36.36	30	421.71	36.36	2789	2789	41.89	0.00	2790	30.02	0.04
15	23	178.54	53.33	23	405.25	53.33	1903	1906	78.43	0.16	1906	38.01	0.16
11	22	341.59	100.00	18	59.33	63.54	1330	1330	151.70	0.00	1330	33.52	0.00
38	38	31.35	0.00	38	3.25	0.00	4975	4975	21.34	0.00	4975	22.10	0.00
32	34	66.96	6.25	32	319.98	0.00	4389	4389	57.22	0.00	4389	12.11	0.00
18	24	218.75	33.33	24	254.78	33.33	2682	2688	89.97	0.22	2682	94.84	0.00
13	22	514.81	69.23	23	217.25	76.92	1962	1962	188.12	0.00	1964	51.47	0.10
9	20	435.86	122.22	16	64.92	77.78	1405	1405	197.89	0.00	1405	40.78	0.00
30	30	41.63	0.00	30	2.55	0.00	4923	4923	28.49	0.00	4923	32.00	0.00
29	30	98.13	3.45	30	85.17	3.45	4772	4772	59.34	0.00	4772	63.00	0.00
15	22	326.79	46.67	23	187.53	53.33	2782	2787	108.96	0.18	2782	36.05	0.00
11	22	566.68	100.00	20	53.91	81.82	1970	1977	199.76	0.36	1976	190.64	0.30
30	30	51.96	0.00	30	9.33	0.00	5089	5089	34.11	0.00	5089	37.20	0.00
27	29	103.40	7.41	28	35.19	3.70	5009	5009	70.88	0.00	5009	9.30	0.00
15	23	517.40	53.33	23	27.88	53.33	3002	3002	213.44	0.00	3018	211.21	0.53
29	29	80.46	0.00	29	1.62	0.00	5364	5364	49.46	0.00	5364	3.19	0.00
23	25	163.18	8.70	24	34.36	4.35	4772	4772	65.12	0.00	4772	6.20	0.00
13	21	699.29	61.54	22	51.40	69.23	2992	2992	241.33	0.00	2995	299.70	0.10
Avrg.	42.00	121.96	28.39	40.93	90.01	22.58	Avrg.	3066.60	52.37	0.11	3065.25	50.65	0.03
# Best	11	16		18	24		# Best	31	12		33	28	

solution found among these 10 executions. Since the CPU we used and those used by other authors are different (AMD Semprom 1.6 Stanimirovic et al., 2007 and pentium III 0.8 Domínguez-Marín et al., 2005), average running times were scaled down by the appropriate factors according to published standards. (See CPU speed benchmarks at http://www.spec.org/cpu2000/ in Table 11).

In the following we describe, in an exhaustive way, our computational results. We report detailed results on the execution of MOD-VNS for the 8 types of λ-parameters on the 40 *N-median* instances of the ORLIB. We also compare our results with those available from the papers by Domínguez-Marín et al. (2005) and Stanimirovic et al. (2007). Problems T1 and T4 were reported on both papers, whereas results for problems T2, T3 and T5–T8 are only available from the second paper. Nevertheless, we are also interested in comparing our new implementation of VNS with the one in Domínguez-Marín et al. (2005). Therefore, we report the comparisons in different formats. Tables 12 and 14 include the results of the VNS in Domínguez-Marín et al. (2005), the HGA1 given in Stanimirovic et al. (2007) and our results obtained by MOD-VNS. In addition, in Tables 13, 15 and 16 we report on the comparison of MOD-VNS and the genetic algorithm HGA1 in Stanimirovic et al. (2007). In order to compare the quality of our results, we use the best known solutions found that have been reported either by Domínguez-Marín et al. (2005) and/or Stanimirovic et al. (2007), with the exception of problems' type T1 where we use those reported by Beasley in Beasley (1990).

To assess the performance of our VNS, we show comparative results on these benchmark instances. The main criteria used for comparisons are the ones commonly used in the literature: gap and computing times. Consequently and like many previous studies, we make our comparisons based on information such as num-

Table 14
Results for **T4** (k_1,k_2)-trimmed mean.

	HGA1			VNS			MOD-VNS		
BEST	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)
4523	4523	1.58	0.00	4523	1.27	0.00	4523	0.00	0.00
2987	2987	2.02	0.00	2987	3.80	0.00	2987	0.00	0.00
3067	3067	1.91	0.00	3074	2.80	0.23	3067	0.29	0.00
2137	2137	2.59	0.00	2142	6.98	0.23	2137	0.14	0.00
818	818	3.06	0.00	818	8.22	0.00	818	0.00	0.00
6064	6064	4.25	0.00	6079	7.88	0.25	6064	0.11	0.00
4206	4206	6.53	0.00	4206	13.41	0.00	4206	0.00	0.00
3182	3182	8.13	0.00	3182	28.30	0.00	3182	0.00	0.00
1807	1807	8.75	0.00	1816	66.39	0.50	1807	0.00	0.00
818	823	11.63	0.61	829	75.91	1.34	818	29.95	0.00
5979	5979	7.45	0.00	5979	13.30	0.00	5979	0.00	0.00
5021	5021	10.21	0.00	5021	25.86	0.00	5021	0.00	0.00
3133	3133	12.31	0.00	3133	97.80	0.00	3133	17.48	0.00
1946	1950	29.76	0.21	1957	303.64	0.57	1946	25.82	0.00
1129	1134	46.68	0.44	1133	415.80	0.35	1129	119.90	0.00
6341	6341	11.53	0.00	6341	24.13	0.00	6341	0.01	0.00
5381	5381	18.54	0.00	5413	43.83	0.59	5381	1.46	0.00
3437	3437	28.00	0.00	3443	261.86	0.17	3437	9.35	0.00
1921	1926	30.65	0.26	1933	779.77	0.62	1921	103.50	0.00
1146	1150	101.60	0.35	1152	1108.48	0.52	1150	125.90	0.35
7245	7245	15.16	0.00	7245	24.22	0.00	7245	0.01	0.00
6685	6685	34.91	0.00	6722	58.58	0.55	6685	0.00	0.00
3306	3307	42.84	0.03	3306	639.95	0.00	3306	13.32	0.00
2002	2004	64.45	0.10	2005	1455.81	0.15	2004	258.05	0.10
1148	1148	244.77	0.00	1151	2552.02	0.26	1148	274.44	0.00
7787	7787	20.37	0.00	7787	48.11	0.00	7787	1.51	0.00
6444	6444	37.62	0.00	6444	141.70	0.00	6444	0.01	0.00
3200	3202	76.60	0.06	3210	1113.89	0.31	3200	118.80	0.00
2004	2005	107.39	0.05	2006	3178.69	0.10	2004	258.10	0.00
1290	1296	329.52	0.47	1308	4942.75	1.40	1290	276.23	0.00
8044	8046	31.50	0.02	8046	66.16	0.02	8046	53.80	0.02
7277	7278	44.82	0.01	7280	162.97	0.04	7277	234.60	0.00
3403	3403	108.81	0.00	3413	2377.72	0.29	3403	221.48	0.00
2023	2033	233.71	0.49	2023	5657.56	0.00	2023	247.70	0.00
8191	8191	37.22	0.00	8191	72.58	0.00	8191	1.01	0.00
7796	7796	101.99	0.00	7820	201.64	0.31	7796	2.36	0.00
3598	3600	181.60	0.06	3604	3170.70	0.17	3600	199.01	0.06
8720	8720	41.90	0.00	8720	140.84	0.00	8720	0.00	0.00
7360	7360	70.55	0.00	7360	313.03	0.00	7360	0.00	0.00
3710	3710	316.02	0.00	3718	5422.73	0.22	3715	295.32	0.13
Avrg.	4158.15	57.86	0.08	4163.00	578.01	0.23	4157.28	72.24	0.02
# Best	26	14		17	0		35	26	

ber of best solutions found, CPU time and gap obtained by each algorithm.

Since our test problems for all problem types are based on the same 40 *N*-median instances of the ORLIB, we only include their names and sizes, i.e. number of nodes (*M*) and number of facilities to be located (*N*), in our first table, namely Table 12. We also remark that the last two rows in all tables report, respectively, the averages of the corresponding columns (one to the last row) and the number of times that the best result (either CPU time or objective function value) is obtained in that column (last row).

Tables 12 and 14 present the detailed computational results of MOD-VNS as well as those obtained by VNS and HGA1, over the benchmark instances for the problem types T1 and T4. In these tables, we indicate the optimal or best known objective function values in the columns (*OPT*) in Table 12 and (*BEST*) in Table 14. The remaining columns show the objective values (*OBJ*), the running times in seconds (*T*) and the gap (*GAP*) obtained by each algorithm. (The reader should observe that 0.00 in columns *T* means that the CPU time was less that 0.01 seconds.)

Table 12 reports the results of MOD-VNS algorithm applied to problem type T1. We observe that the number of optimal solutions found are 31 out of 40 (31/40) for MOD-VNS, 23 out of 40 (23/40) for HGA1 and 17 out of 40 (17/40) for VNS. MOD-VNS takes clear advantage with respect to VNS in all rates, there is a tradeoff between quality of the solution and computing time required to obtain this solution: the average time is equal to 493.63 scaled seconds for VNS and 73.17 scaled seconds for MOD-VNS (less than 15% of the required running time). Notice that the maximal computing time required by MOD-VNS does not exceed 335 seconds, whereas the maximum time required by VNS was almost 4774 seconds.

The reader may also observe that the quality of the solutions provided by MOD-VNS (gap of 0.03%, on average) is comparable, although slightly better than that of HGA1 (0.12%, on average). Nevertheless, running time required by MOD-VNS is larger on

Table 15
Results for **T5** and **T6**.

T5							**T6**						
	HGA1			MOD-VNS				HGA1			MOD-VNS		
BEST	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)	*BEST*	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)
2941	2941	1.56	0.00	2941	0.00	0.00	2873	2873	1.67	0.00	2873	0.00	0.00
2075	2075	2.12	0.00	2075	0.35	0.00	2011	2018	2.02	0.35	2011	0.52	0.00
2160	2160	2.13	0.00	2160	0.50	0.00	2062	2062	1.85	0.00	2062	0.66	0.00
1532	1537	2.28	0.33	1532	2.36	0.00	1491	1497	2.44	0.40	1491	2.58	0.00
689	691	3.25	0.29	689	2.85	0.00	662	662	2.90	0.00	662	0.34	0.00
3938	3938	4.57	0.00	3938	0.00	0.00	3884	3884	4.25	0.00	3884	0.00	0.00
2835	2835	8.70	0.00	2835	0.00	0.00	2795	2795	6.62	0.00	2795	0.00	0.00
2244	2244	6.31	0.00	2244	0.00	0.00	2190	2190	7.50	0.00	2190	1.65	0.00
1374	1380	9.05	0.44	1374	182.61	0.00	1343	1349	10.68	0.45	1343	13.52	0.00
632	633	10.60	0.16	632	148.48	0.00	619	619	8.94	0.00	619	34.45	0.00
3869	3869	7.74	0.00	3869	0.00	0.00	3827	3827	7.32	0.00	3827	0.00	0.00
3344	3344	15.60	0.00	3344	0.21	0.00	3288	3288	12.64	0.00	3288	0.00	0.00
2198	2198	15.48	0.00	2198	5.83	0.00	2173	2173	14.76	0.00	2173	18.56	0.00
1490	1493	17.87	0.20	1490	11.08	0.00	1474	1479	21.12	0.34	1474	99.28	0.00
870	870	29.21	0.00	870	45.59	0.00	861	861	29.59	0.00	861	87.43	0.00
4094	4094	12.93	0.00	4094	0.44	0.00	4065	4065	15.65	0.00	4065	0.27	0.00
3512	3512	23.84	0.00	3512	2.71	0.00	3487	3487	23.26	0.00	3487	0.00	0.00
2414	2415	30.58	0.04	2414	215.99	0.00	2388	2388	33.90	0.00	2388	96.41	0.00
1430	1430	55.44	0.00	1430	282.61	0.00	1418	1420	47.16	0.14	1420	82.88	0.14
905	909	48.75	0.44	908	281.46	0.33	895	895	93.08	0.00	895	86.17	0.00
4578	4578	15.10	0.00	4578	0.00	0.00	4560	4560	15.57	0.00	4560	0.00	0.00
4310	4310	35.20	0.00	4310	0.00	0.00	4269	4269	27.58	0.00	4269	2.83	0.00
2317	2319	48.96	0.09	2317	46.47	0.00	2296	2303	62.13	0.30	2296	164.44	0.00
1485	1485	86.15	0.00	1485	90.43	0.00	1477	1477	84.43	0.00	1477	71.11	0.00
924	924	164.65	0.00	924	178.05	0.00	914	914	211.89	0.00	914	101.81	0.00
4970	4970	25.52	0.00	4970	1.47	0.00	4945	4945	22.77	0.00	4945	0.00	0.00
4162	4162	44.95	0.00	4162	11.99	0.00	4143	4143	42.61	0.00	4143	0.00	0.00
2255	2257	86.45	0.09	2255	87.70	0.00	2243	2243	82.71	0.00	2243	233.23	0.00
1524	1530	148.33	0.39	1527	154.92	0.20	1515	1517	152.67	0.13	1520	154.70	0.33
1009	1009	357.46	0.00	1010	309.70	0.10	1000	1000	216.69	0.00	1000	212.13	0.00
5054	5054	31.54	0.00	5054	1.42	0.00	5031	5031	30.25	0.00	5031	0.00	0.00
4683	4683	54.65	0.00	4683	3.16	0.00	4610	4610	48.91	0.00	4610	0.00	0.00
2356	2360	125.45	0.17	2357	235.5	0.04	2346	2346	116.57	0.00	2346	120.63	0.00
1517	1517	288.22	0.00	1518	225.3	0.07	1511	1511	310.53	0.00	1515	311.20	0.26
5209	5209	34.15	0.00	5209	0.001	0.00	5189	5189	36.08	0.00	5189	0.00	0.00
4981	4981	80.25	0.00	4981	4.98	0.00	4953	4953	101.03	0.00	4953	7.24	0.00
2540	2540	198.20	0.00	2540	240.20	0.00	2522	2522	240.44	0.00	2525	152.34	0.12
5542	5542	40.97	0.00	5542	0.00	0.00	5518	5518	46.78	0.00	5518	1.62	0.00
4745	4745	58.95	0.00	4745	0.00	0.00	4678	4678	66.50	0.00	4678	0.00	0.00
2578	2578	344.49	0.00	2582	319.30	0.16	2566	2566	211.50	0.00	2567	213.50	0.04
Avrg.	2783.03	59.92	0.07	2782.45	77.34	0.02	Avrg.	2753.18	57.54	0.05	2752.68	56.51	0.02
# Best	30	13		34	27		*# Best*	33	14		35	26	

average than the running time for HGA1 although in 26 out of the 40 (26/40) instances MOD-VNS required less CPU time than HGA1 to find the best solution. This shows a similar behavior of both algorithms on problems of type T1.

Table 14 reports the comparisons, in the same format as above, for problem type T4 ((k_1,k_2)-trimmed mean problem). Optimal solutions for these problems on *N*-median instances are not known so far, so we considered the best known solutions reported in the literature. MOD-VNS reached 35 out of 40 (35/40), HGA1 26 out of 40 (26/40) and VNS 17 out of 40 (17/40) of the best-known solutions. The behavior of MOD-VNS applied to T4 is quite similar to T1. We can observe that after a reasonable computing time MOD-VNS and HGA1 solve large (k_1,k_2)-trimmed mean problems. The computing time required by HGA1 (57.86 seconds, on average) is shorter than the time needed by MOD-VNS (72.24 seconds, on average) although as observed before in Table 12 for the problem type T1, MOD-VNS requires in 26 out of 40 instances less CPU time than HGA1 to find the best solution. Moreover, MOD-VNS provides higher number of best-known solutions than HGA1.

By analyzing Table 13, we observe that the average gap over T2 (*N*-center) is about 28% for HGA1 and 22% for MOD-VNS. Note that this is the worst gap among all problem types (T1–T8). Nevertheless, in a number of instances, the best known objective function value was attained, 18 out of 40 (18/40) for MOD-VNS as compared to only 11 out of 40 (11/40) for HGA1. Concerning the CPU time, we point out that MOD-VNS is about 27% faster than HGA1 for this type of problem. Turning to problem type T3, we observe that the performance of both algorithms is similar, although MOD-VNS is slightly better both in CPU time (3% faster) and in quality of solutions (0.03% gap for MOD-VNS as compared to 0.11% gap for HGA1).

Table 16
Results for **T7** and **T8**.

T7							**T8**						
	HGA1			MOD-VNS				HGA1			MOD-VNS		
BEST	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)	*BEST*	*OBJ*	*T*	*GAP* (%)	*OBJ*	*T*	*GAP* (%)
3924	3924	1.57	0.00	3924	0.00	0.00	1986	1986	1.57	0.00	1986	0.00	0.00
2769	2769	2.45	0.00	2769	0.41	0.00	1400	1400	2.61	0.00	1400	1.33	0.00
2874	2874	2.06	0.00	2874	2.69	0.00	1456	1456	2.12	0.00	1456	0.00	0.00
2054	2061	2.10	0.34	2054	0.31	0.00	1038	1040	2.46	0.19	1038	3.14	0.00
923	923	3.21	0.00	923	4.95	0.00	468	468	3.49	0.00	468	2.45	0.00
5250	5250	4.08	0.00	5250	0.00	0.00	2642	2642	4.27	0.00	2642	0.00	0.00
3778	3778	6.40	0.00	3778	0.00	0.00	1902	1902	6.60	0.00	1902	0.00	0.00
2993	2993	7.25	0.00	2993	0.00	0.00	1516	1516	7.58	0.00	1516	18.96	0.00
1834	1839	8.63	0.27	1839	8.09	0.27	924	924	10.83	0.00	924	27.18	0.00
844	844	10.15	0.00	845	30.08	0.12	424	424	11.37	0.00	424	11.89	0.00
5155	5155	7.33	0.00	5155	0.00	0.00	2588	2588	8.04	0.00	2588	0.00	0.00
4450	4450	11.58	0.00	4450	0.00	0.00	2249	2249	10.35	0.00	2249	0.00	0.00
2936	2936	13.46	0.00	2940	13.73	0.14	1474	1474	16.44	0.00	1474	34.74	0.00
1988	1993	19.69	0.25	2012	26.64	1.21	998	998	23.11	0.00	998	214.21	0.00
1161	1161	23.55	0.00	1161	173.10	0.00	581	584	27.55	0.52	583	245.09	0.34
5456	5456	11.70	0.00	5456	0.00	0.00	2735	2735	13.80	0.00	2735	0.00	0.00
4681	4681	26.69	0.00	4681	16.19	0.00	2347	2347	25.46	0.00	2347	8.83	0.00
3222	3226	24.77	0.12	3224	12.81	0.06	1618	1619	24.20	0.06	1620	10.81	0.12
1890	1913	44.05	1.22	1890	134.60	0.00	950	960	54.06	1.05	950	239.39	0.00
1192	1198	45.77	0.50	1192	113.40	0.00	604	607	83.68	0.50	607	30.47	0.50
6106	6106	16.46	0.00	6106	0.00	0.00	3062	3062	17.04	0.00	3062	0.00	0.00
5745	5745	35.11	0.00	5745	0.00	0.00	2888	2888	38.23	0.00	2888	0.00	0.00
3078	3091	41.89	0.42	3078	141.50	0.00	1554	1555	41.00	0.06	1554	31.82	0.00
1981	1984	69.82	0.15	1981	143.01	0.00	986	1000	97.79	1.42	986	106.07	0.00
1224	1226	130.82	0.16	1226	126.34	0.16	619	619	227.01	0.00	619	225.05	0.00
6628	6628	24.47	0.00	6628	0.97	0.00	3323	3323	22.75	0.00	3323	0.47	0.00
5556	5556	45.97	0.00	5556	5.00	0.00	2784	2784	48.13	0.00	2784	0.00	0.00
3007	3011	89.56	0.13	3008	110.02	0.03	1508	1512	101.77	0.27	1508	113.80	0.00
2033	2035	113.72	0.10	2035	115.34	0.10	1022	1022	188.39	0.00	1022	190.03	0.00
1336	1340	256.26	0.30	1336	153.50	0.00	683	683	584.48	0.00	685	269.07	0.29
6735	6735	24.78	0.00	6735	0.00	0.00	3374	3374	31.09	0.00	3374	8.33	0.00
6226	6226	46.54	0.00	6226	0.00	0.00	3143	3143	45.45	0.00	3143	9.33	0.00
3146	3146	112.38	0.00	3146	117.33	0.00	**1572**	1582	120.71	0.64	1572	118.10	0.00
2024	2024	216.48	0.00	2024	143.40	0.00	1019	1023	338.46	0.39	1021	356.05	0.20
6944	6944	35.51	0.00	6944	0.00	0.00	3479	3479	33.89	0.00	3479	0.00	0.00
6639	6639	67.82	0.00	6639	0.00	0.00	3330	3330	91.08	0.00	3330	5.58	0.00
3383	3383	156.16	0.00	3390	162.02	0.21	1700	1700	235.65	0.00	1707	245.01	0.41
7390	7390	49.03	0.00	7390	7.88	0.00	3704	3704	50.37	0.00	3704	0.00	0.00
6310	6310	68.42	0.00	6310	0.00	0.00	3187	3187	65.54	0.00	3187	0.00	0.00
3434	3434	197.24	0.00	3434	142.67	0.00	1722	1722	240.43	0.00	1730	356.02	0.46
Avrg.	3709.43	48.24	0.10	3708.68	46.92	0.06	Avrg.	1865.28	71.57	0.13	1864.63	69.19	0.06
# Best	28	14		32	26		# Best	32	15		33	25	

From Tables 15 and 16, we observe that MOD-VNS and HGA1 also compare similarly on problem types T5, T6, T7 and T8. On the one hand, CPU time required by MOD-VNS is slightly smaller than the one needed by HGA1 on problems T6, T7 and T8 and slightly larger for T5; although in that case this excess is always less than 24%. On the other hand, the quality of solutions found by MOD-VNS is superior than the one obtained by HGA1, because the average gap provided by HGA1 over all instances is higher than that given by MOD-VNS. Moreover, as in all the previously commented problem types MOD-VNS reaches a higher number of best-known (optimal) objective function values than HGA1.

It should be emphasized that the performance of both procedures is rather good on all problem types except for type T2 (i.e., *N*-center problems). For problem T2 the average gap is relatively large. In spite of that, the quality of the solutions obtained by MOD-VNS for problems of type T2 is better than that provided by HGA1.

We show in Fig. 1 a comparison of the performance of algorithm HGA1 versus MOD-VNS in terms of % gap (x-axis) and CPU time (y-axis) for problem types T1 and T3–T8. The reader should observe that *N*-center problems, namely problem type T2, are excluded from this comparative since its average gap for both algorithms is above 20%, some orders of magnitude greater than the rest. Thus, we have decided not to include it because the representation of those data on the same figure would have perturbed the visualization of the comparison of the remaining ones. From this table, we see that for problem types T3, T6, T7 and T8, MOD-VNS requires less computing time than HGA1 whereas the contrary occurs for problems T1, T4 and T5. However, in all problem types the quality of solutions obtained by MOD-VNS is higher than the quality of solutions obtained by HGA1 since MOD-VNS always gets gaps with respect to best known solutions smaller than the corresponding ones obtained by HGA1.

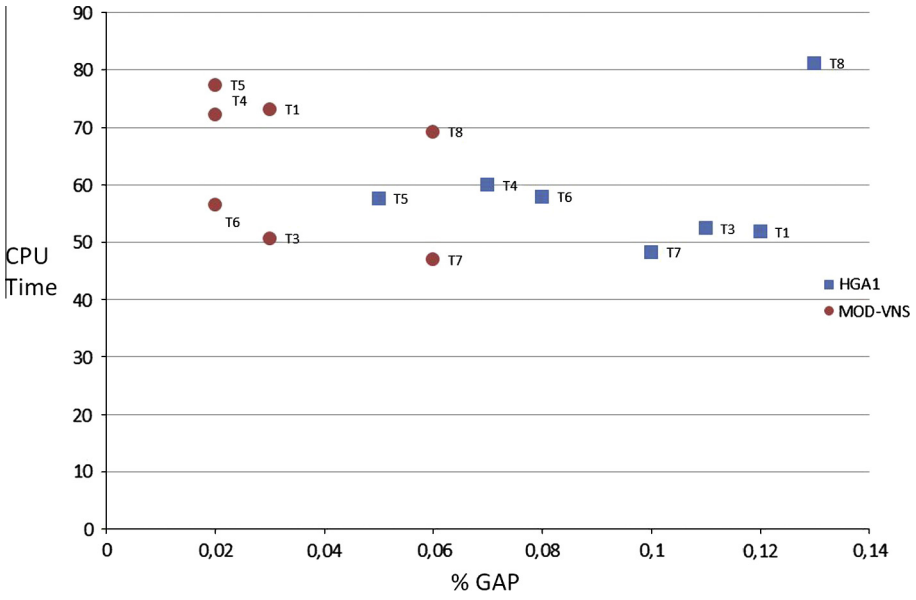

Fig. 1. GAP-CPU Time chart of the performance of algorithms HGA1 and MOD-VNS.

From the above analysis, we observe that MOD-VNS performs better than HGA1 in terms of the quality of the solutions found (smaller gaps, higher number of best solutions found). Concerning average CPU times the performance of both algorithms is similar: in five out of eight problem types, namely T2, T3, T6, T7 and T8, MOD-VNS is slightly faster whereas the opposite occurs for T1, T4 and T5. Nevertheless, in all problem types the number of instances with less required CPU time is always higher with MOD-VNS than with HGA1.

Finally, we would like to point out that the new encoding of solutions, specifically developed for applying VNS algorithms to the DOMP problem, yields results considerably better than previous encodings as observed comparing VNS and MOD-VNS when it is possible.

5. Conclusions

In this paper, we present a modified VNS heuristic, named MOD-VNS (there exists another VNS for the same problem in the literature) for the Discrete Ordered Median Problem (DOMP). This algorithm is based on refined neighborhoods' structures that favor faster improvement of the objective function in the local search phase and allow an efficient encoding of the solutions of the DOMP avoiding sorting in the evaluation of the objective function at each considered solution. Comprehensive computational experiments, on ORLIB *N*-median instances, demonstrate the robustness of the proposed algorithm with respect to solution quality. For five problem types, namely T2, T3, T6, T7 and T8, running times for MOD-VNS are better on average than the corresponding running times obtained by HGA1; moreover in all problem types the number of instances for which MOD-VNS requires less computing time is greater than the corresponding one for HGA1. In spite of that, for 3 problem types, T1, T4 and T5, the average running times of our algorithm are higher than HGA1. Comparisons with results from the literature show the appropriateness of applying the proposed algorithm. Computational results show that in many cases MOD-VNS outperforms other existing algorithms for these problems in number of best solution found, average gap and CPU times.

Acknowledgements

This research has been supported by the Spanish Ministry of Science and Innovation under Grant MTM2010-19576-C02-01, by the Junta de Andalucía (Spain)/FEDER under Grant FQM5849 and by Gobierno de Canarias Grant SolSubC200801000048.

References

Beasley, J. (1990). OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society, 41*, 1069–1072. <http://people.brunel.ac.uk/mastjjb/jeb/info.html>.

Boland, N., Domínguez-Marín, P., Nickel, S., & Puerto, J. (2006). Exact procedures for solving the discrete ordered median problem. *Computers and Operations Research, 33*, 3270–3300.

Daskin, M. (1995). *Network and discrete location: Models, algorithms, and applications*. Wiley.

Domínguez-Marín, P. (2003). *The Discrete ordered median problem: Models and solution methods*. PhD thesis, University of Kaiserslautern.

Domínguez-Marín, P., Nickel, S., Hansen, P., & Mladenović, N. (2005). Heuristic procedures for solving the discrete ordered median problem. *Annals of Operations Research, 136*, 145–173.

Drezner, Z., & Hamacher, H. (2002). *Facility location: Applications and theory*. Berlin Heidelberg New York: Springer.

Hansen, P., & Mladenović, N. (1997). Variable neighborhood search for the *p*-median. *Location Science, 5*(4), 207–226.

Hansen, P., & Mladenović, N. (2001). Developments of variable neighborhood search. In C. C. Ribeiro & P. Hansen (Eds.), *Essays and surveys in metaheuristics* (pp. 415–439). Kluwer Academic Publishers.

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research, 130*(3), 449–467.

Hansen, P., & Mladenović, N. (2003). Variable neighborhood search. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics. International series in operations research & management science* (Vol. 57). Boston, MA: Kluwer Academic Publishers.

Hansen, P., Mladenović, N., & Moreno-Pérez, J. A. (2010). Variable neighborhood search: Methods and applications. *Annals of Operations Research, 175*, 367–407.

Hansen, P., Mladenović, N., & Pérez-Brito, D. (2001). Variable neighborhood decomposition search. *Journal of Heuristics, 7*, 335–350.

Kalcsics, J., Nickel, S., Puerto, J., & Rodríguez-Chía, A. M. (2010a). The ordered capacitated facility location problem. *TOP, 18*(1), 203–222.

Kalcsics, J., Nickel, S., Puerto, J., & Rodríguez-Chía, A. M. (2010b). Distribution systems design with role dependent objectives. *European Journal of Operational Research, 202*(2), 491–501.

Kariv, O., & Hakimi, L. (1979). An algorithmic approach to network location problems. II: The *p*-medians. *SIAM Journal on Applied Mathematics, 37*, 539–560.

Marín, A., Nickel, S., Puerto, J., & Velten, S. (2009). A flexible model and efficient solution strategies for discrete location problems. *Discrete Applied Mathematics, 157*(5,6), 1128–1145.

Marín, A., Nickel, S., & Velten, S. (2010). An extended covering model for flexible discrete and equity location problems. *Mathematical Methods of Operations Research, 71*(1), 125–163.
Mirchandani, P., & Francis, R. (1990). *Discrete location theory*. New York, NY: Wiley.
Mladenović, N., Labbé, M., & Hansen, P. (2003). Solving the p-center problem with tabu search and variable neighborhood search. *Networks, 42*(1), 48–64.
Nickel, S. (2001). Discrete ordered Weber problems. In *Operations research proceedings 2000* (pp. 71–76). Springer Verlag.
Nickel, S., & Puerto, J. (2005). *Facility location – A unified approach*. Springer Verlag.
Nickel, S., & Puerto, J. (1999). A unified approach to network location problems. *Networks, 34*, 283–290.
Puerto, J., Ramos, A. B., & Rodríguez-Chía, A. M. (2011). Single-allocation ordered median hub location problems. *Computers and Operations Research, 38*, 559–570.
Resende, M. G. C., & González-Velarde, J. L. (2003). GRASP: Procedimientos de búsqueda miopes aleatorizados y adaptativos. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 19*, 61–76.
Rodríguez-Chía, A., Nickel, S., Puerto, J., & Fernández, F. (2000). A flexible approach to location problems. *Mathematical Methods of Operations Research, 51*, 69–89.
Stanimirovic, Z., Kratica, J., & Dugosija, D. (2007). Genetic algorithms for solving the discrete ordered median problem. *European Journal of Operational Research, 182*, 983–1001.